

TIPY PRIMITIVI

- I tipi primitivi sono tipi di dato non complessi fissati dalle specifiche del linguaggio.
 - Posso manipolare i tipi primitivi utilizzando gli operatori.
 - Le variabili contengono direttamente il dato.
- In ActionScript i tipi primitivi sono **int**, **uint**, **Number**, **Boolean**, **String** a questi si aggiungo due tipi *speciali*: **void** e **null**.

int

- Il tipo di dati **int** memorizza un numero intero con segno utilizzando 32 bit (4 byte). Il valore può andare da **-2,147,483,648** (-2^{31}) a **2,147,483,647** ($2^{31} - 1$) inclusi.
- Il valore predefinito di una variabile di tipo **int** dichiarata ma non inizializzata è **0**.

uint

- Il tipo **uint** memorizza un numero intero positivo utilizzando 32 bit (4 byte).
- Il valore può andare da da 0 a 4,294,967,295 ($2^{32} - 1$) inclusi.
- Il valore predefinito di una variabile di tipo **uint** dichiarata ma non inizializzata è **0**.

Number

- Il tipo **Number** può rappresentare numeri interi, numeri interi senza segno e numeri a virgola mobile utilizzando 64 bit (8 byte).
- La conversione tra numero intero e numero decimale è automatica: per forzare **Number** a rappresentare un numero decimale, bisogna includere il punto decimale (Ad esempio: **123.0**)
- Se il numero è impostato come intero i risultati delle operazioni vengono arrotondate al numero intero più vicino.

Number

- Il valore Massimo e il valore minimo che il tipo *Number* è in grado di contenere sono memorizzati nelle proprietà statiche **MAX_VALUE** e **MIN_VALUE** della classe *Number*.
- **Number.MAX_VALUE == 1.79769313486231e+308**
- **Number.MIN_VALUE == 4.940656458412467e-324**
- La notazione qui utilizzata è quella scientifica (o esponenziale) ed equivale (nel primo caso) a $1.79769313486231 * 10^{308}$, cioè 179,769,313,486,231 seguito da 294 zeri.

Number

- Questa possibilità di rappresentare numeri estremamente grandi ed estremamente piccoli viene pagata in termini di precisione nei calcoli. Si avrà una buona approssimazione per i calcoli su numeri relativamente vicini allo zero, mentre calcoli su numeri molto grandi o molto piccoli saranno molto approssimati.
- Quando viene utilizzato per memorizzare i numeri interi il tipo **Number** è in grado di gestire valori da **-9,007,199,254,740,992** (-2^{53}) a **9,007,199,254,740,992** ($2^{53}-1$) compresi.
- Il valore di default di una variabile **Number** è **NaN**.

Boolean

- Il tipo di dati Boolean può avere due valori simbolici **true** e **false**. Nessun altro valore è consentito per le variabili di questo tipo.
- Il valore predefinito di una variabile booleana dichiarata ma non inizializzata è **false**.
- Astrattamente il valore booleano è rappresentato da un unico bit.

String

- Il tipo di dati **String** rappresenta una sequenza di caratteri a 16 bit che può includere lettere, numeri e segni di punteggiatura.
- Le stringhe vengono memorizzate come caratteri Unicode, utilizzando il formato UTF-16.
- Un'operazione su un valore **String** restituisce una nuova istanza della stringa.

null

- Il tipo di dati **null** contiene un unico valore: la costante **null**;
- **null** è il valore che contengono le variabili di tipo **string** e tutte le variabili complesse quando sono state create, ma non è stato loro assegnato alcun valore.

DICHIARAZIONE DI VARIABILI

- Per dichiarare una variabile in ActionScript si usa la parola riservata **var** seguita dal nome della variabile, dai due punti e dal tipo:

```
var pippo:String;
```

- Opzionalmente si può assegnare un valore alla variabile all'atto della dichiarazione (inizializzazione):

```
var pippo:String = "Hello World" ;
```

DICHIARAZIONE DI VARIABILI DI TIPI PRIMITIVI

```
//dichiarazioni di variabili in actionscript
/* a conterrà un numero, s una stringa k
   true o false */
var a:Number;
var s:String;
var k:Boolean;
/* per b e messaggio oltre a dichiarare il
   tipo viene impostato un valore iniziale */
var b:Number = 1;
var messaggio:String = "Ciao a tutti" ;
```

TIPI DERIVATI O COMPLESSI

- Per rappresentare dati complessi ho a disposizione alcuni tipi complessi che il linguaggio mi offre oppure ne posso creare ad hoc.
- Per i tipi complessi la variabile contiene il **puntatore** cioè il numero della casella, l'indirizzo in cui il dato è memorizzato sul computer.
- Nei linguaggi orientati agli oggetti il concetto di **tipo** e il concetto di **classe** coincidono.

ESEMPI DI TIPI COMPLESSI

- **Array** rappresenta un tabella di valori. Posso recuperare un valore nella tabella utilizzando l'indice.
- **Object** rappresenta un variabile strutturata che organizza un dato complesso.
- **MovieClip** rappresenta un clip filmato.
- **TextField** rappresenta un campo di testo.
- **MioTipo** un tipo creato dal programmatore.

ARRAY

- Un **Array** è un elenco di elementi recuperabile attraverso un indice.
- Non occorre che gli elementi dell'array abbiano lo stesso tipo di dati. È possibile inserire numeri, dati, stringhe, oggetti, altri array.

USO DEGLI ARRAY

- Gli array possono essere utilizzati in modi diversi.
- Il modo più semplice di creare un Array è assegnare ad una variabile di tipo Array un elenco di valori sotto forma di costante.
- La posizione di un elemento nell'array è detta *indice*. Tutti gli array sono con base zero, ovvero [0] è il primo elemento dell'array, [1] è il secondo, e così via.
- Per identificare un elemento di un Array il nome della variabile viene seguito dall'indice tra parentesi quadre.

LA LEGGIBILITÀ DEL CODICE

Leggibilità

- Scrivere programmi *sensati e leggibili* è difficile, ma molto importante
- È essenziale per lavorare in gruppo
- Aiuto il debugging
- Aiuta a riutilizzare il codice e quindi ci risparmia fatica

Leggibilità significa:

- Progettare con chiarezza
- Scrivere codice con chiarezza

Progettare con chiarezza

- Dedicare il tempo necessario alla progettazione della nostra applicazione non è tempo perso.
- Ci aiuterà a chiarire la logica e la sintassi del nostro lavoro.
- Più avremo sviluppato l'algoritmo che sta alla base della nostra applicazione più il nostro programma sarà comprensibile

Scrivere con chiarezza

- La chiarezza della scrittura si ottiene attraverso due *tecniche* :
- L'***indentazione***: inserire spazi o tabulazioni per mettere subito in evidenza le gerarchie sintattiche del codice.
- I ***commenti***: inserire note e spiegazione nel corpo del codice.

Identazione: un esempio

- Prendiamo in esame questo brano di codice HTML :

```
<table> <tr> <td>a</td> <td>b</td> <td>c</td>
</tr> <tr> <td> <table> <tr> <td>a1</td> </tr>
<tr> <td>a2</td> </tr> </table> </td> <td>b1</td>
<td>c1</td> </tr> </table>
```

Identazione: un esempio

- E confrontiamolo con questo:

```
<table>
  <tr>
    <td>a</td>
    <td>b</td>
    <td>c</td>
  </tr>
  <tr>
    <td>
      <table>
        <tr>
          <td>a1</td>
        </tr>
        <tr>
          <td>a2</td>
        </tr>
      </table>
    </td>
    <td>b1</td>
    <td>c1</td>
  </tr>
</table>
```

Identazione

- Si tratta della stessa tabella, ma nel primo caso ci risulta molto difficile capire come è organizzata. Nel secondo la gerarchia degli elementi risulta molto più chiara.

Identazione

- L'identazione non ha nessun effetto sulla compilazione del programma
- Serve solo a rendere il nostro lavoro più leggibile.

Inserire commenti

- Rende il codice leggibile anche ad altri
- Quando decidiamo di apportare modifiche a cose che abbiamo scritto ci rende la vita più facile.

Delimitatori

- Delimitatori di riga: tutto ciò che segue il contrassegno di commento fino alla fine della riga non viene compilato.
Esempi:

//

- Delimitatori di inizio e fine: tutto ciò compreso tra il contrassegno di inizio e il contrassegno di fine non viene compilato.

/* ... */

<!-- ... -->

INTRODUZIONE ALLA LOGICA

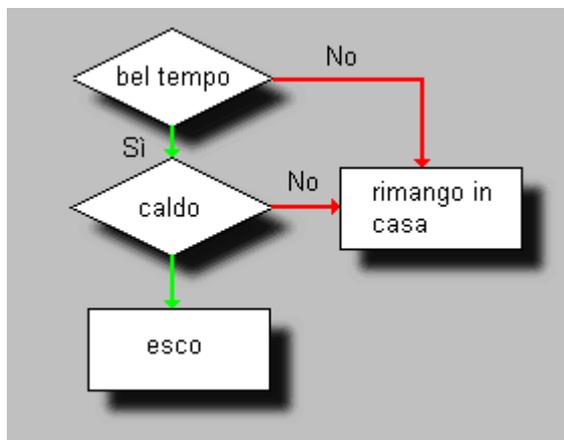
Introduzione

- Nella lezione precedente abbiamo visto che qualsiasi processo logico può essere ricondotto ad una sequenza di eventi elementari (**algoritmo**)
- Che tale sequenza può essere rappresentata con un diagramma di flusso (il quale a sua volta è facilmente traducibile in un particolare programma comprensibile dall'elaboratore).

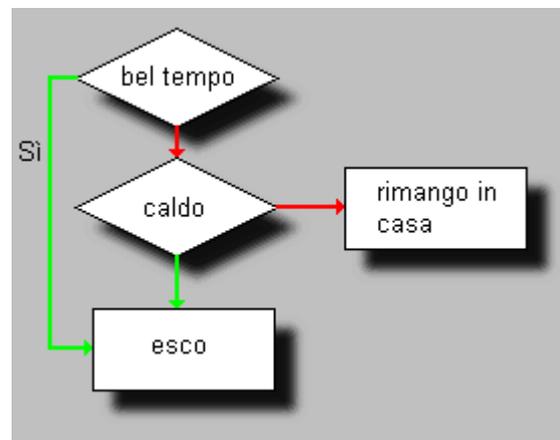
Problema

- Prendiamo questi due enunciati:
 - esco se è bel tempo ed è caldo
 - esco se è bel tempo o se è caldo

Diagrammi di flusso



esco se è bel tempo ed è caldo



esco se è bel tempo o è caldo

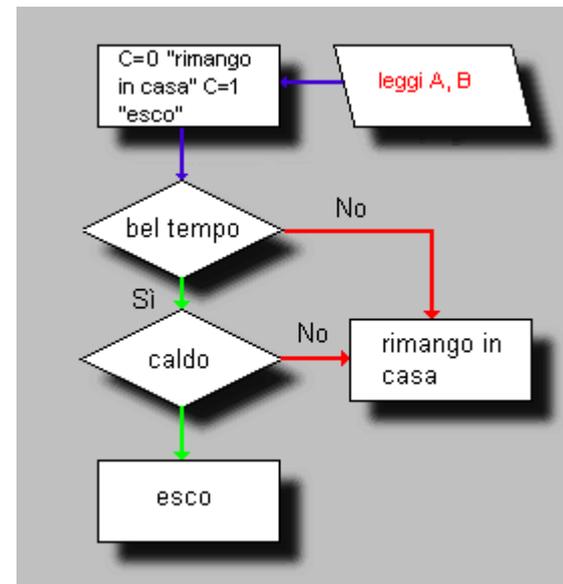
Gli operatori logici

- Come secondo passo, si tratta di convertire i diagramma di flusso in un linguaggio comprensibile dall'elaboratore. Ciò si ottiene con i cosiddetti operatori logici elementari.

operazione	istruzione	porta logica
controllo	se (if)	
azione	allora {esecuzione azione}	
coniunzione	e (and &&)	AND
separazione	o (or)	OR
negazione	non (not !)	NOT

Formalizzazione

- $A = 1$ corrisponde all'evento "bel tempo"
- $B = 1$ corrisponde all'evento "caldo"
- $C = 1$ corrisponde all'azione "esco"
- $A = 0$ corrisponde all'evento "non bel tempo"
- $B = 0$ corrisponde all'evento "non caldo"
- $C = 0$ corrisponde all'azione "resto in casa"



con queste condizioni, il primo diagramma di flusso risulta così formalizzato:

IF A AND B -> C

Gli operatori logici

AND – Congiunzione

falso AND falso	risultato falso
falso AND vero	risultato falso
vero AND falso	risultato falso
vero AND vero	risultato vero

NOT - Negazione

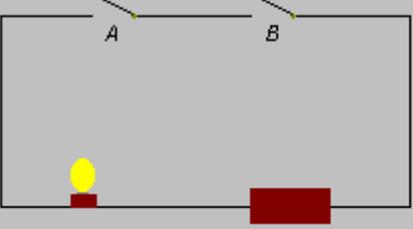
NOT falso	risultato vero
NOT vero	risultato falso

OR - Disgiunzione

falso OR falso	risultato falso
falso OR vero	risultato vero
vero OR falso	risultato vero
vero AND vero	risultato vero

Gli operatori logici

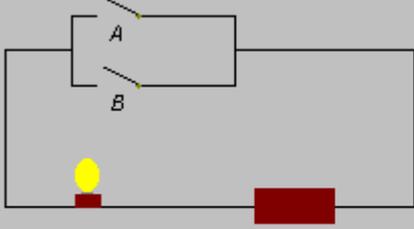
rappresentazione con i circuiti elettrici



se gli interruttori *A* e *B* sono entrambi abbassati, il circuito è chiuso e la lampadina si accende

A	B	C
0	0	0
0	1	0
1	1	1
1	0	0

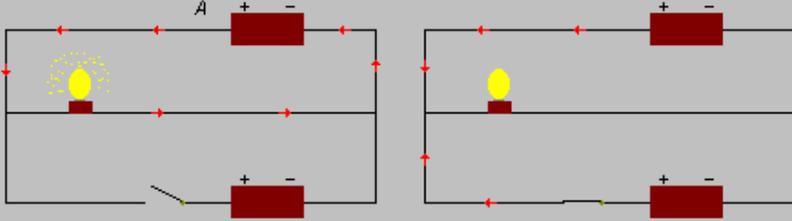
tavola di veridicità per AND



se l'interruttore *A* o *B* è abbassato, il circuito è chiuso e la lampadina si accende.

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

tavola di veridicità per OR



se l'interruttore è aperto, la batteria *A* alimenta il circuito e la lampadina è accesa

se l'interruttore è abbassato, entrambe le batterie alimentano il circuito e la lampadina è spenta

A	C
1	0
0	1

tavola di veridicità per NOT

PROGRAMMAZIONE CONDIZIONALE

Sintassi dell'istruzione if

- L'istruzione if consente di tradurre in un linguaggio di programmazione i ragionamenti fatti parlando della logica Booleana.
- L'istruzione if può avere due forme:
 - if** (espressione) blocco di istruzioni
 - if** (espressione) blocco di istruzioni **else** blocco di istruzioni
- L'espressione che compare dopo la parola chiave **if** deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta vera si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**.
- Per più scelte invece si può usare l'**else if** che permette di porre una condizione anche per le alternative, lasciando ovviamente la possibilità di mettere l'else (senza condizioni) in posizione finale.

Esempio in pseudocodice

```
intero A = 50;
scrivi sullo schermo "Inserisci un numero";
intero B = -numero inserito da tastiera-;
if (B minore di A) {
    scrivi sullo schermo "Il numero inserito è minore di
                          cinquanta";
} else if (B maggiore di A) {
    scrivi sullo schermo "Il numero inserito è maggiore di
                          cinquanta";
} else if (B uguale a A) {
    scrivi sullo schermo "Il numero inserito è cinquanta";
} else {
    //poiché B non è minore, né maggiore né uguale a A
    // A non è un numero
    scrivi sullo schermo "Inserisci un numero";
}
```

Esempio in ActionScript

```
var Number = 50;
var B:Number = input_txt.text;
if (B < A) {
    messaggio_txt.text = "Il numero inserito è minore di
                          cinquanta";
} else if (B > A) {
    messaggio_txt.text = "Il numero inserito è maggiore di
                          cinquanta";
} else if (B == A)
    messaggio_txt.text = "Il numero inserito è cinquanta";
} else { //B non è un numero
    messaggio_txt.text = "Inserisci un numero!!!";
}
```